Webentwicklung

Backend: PHP

Inhalt dieser Einheit

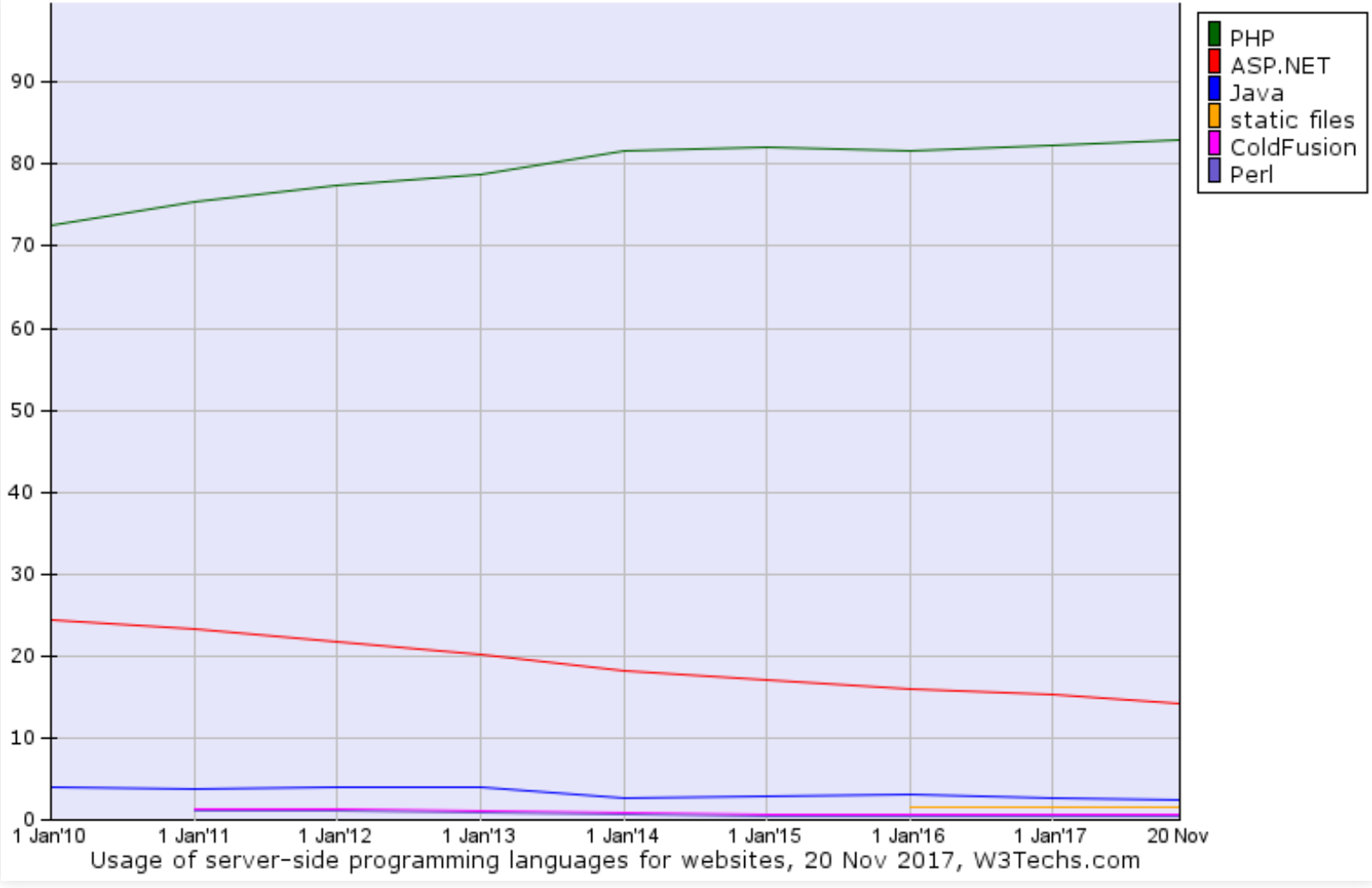
Wdh: HTTP, Webserver, Backend

- Dynamisch auf HTTP-Anfragen reagieren:
  - Grundsätzlich gingen alle Programmiersprachen, über CGI
    - aufrufbar von der Kommandozeile
    - Zugriff auf Umgebungsvariablen
    - Ausgabe in Textform
  - Einige Sprachen bringen aber praktische Dinge mit:
    - HTTP-Header bei Ein- und Ausgabe verarbeiten (z.B. Zugriff auf Eingabe-Werte)
    - Session-Verwaltung
    - Dateisystem- und Datenbankzugriff
- Beispiele:
  - PHP, Java, .Net, Ruby, Python, JavaScript (serverseitig), ...

Backend-Technologie: Kriterien

- **Kosten**
  - Lizenzgebühren vs. freie Nutzung
- **Hosting**
  - leicht vs. aufwändig
- **Community-Größe**
  - für Support (denken Sie an StackOverflow)
- verfügbare **Bibliotheken und Frameworks**
- sonstiger betrieblicher **Kontext**
  - etwa bestehende Systeme
- **Arbeitsmarkt**
  - verfügbare Entwickler/innen

Backend-Technologien: Verbreitung



Quelle: https://w3techs.com/technologies/history_overview/programming_language/ms/y

Bekannte PHP-Systeme

- Facebook
  - (zumindest früher, zumindest teilweise)
- Wikipedia
- Wordpress
- Moodle
- Owncloud (freie Dropbox-Alternative)
- Piwik (freie Google Analytics-Alternative)
- …

Allgemeines zu PHP

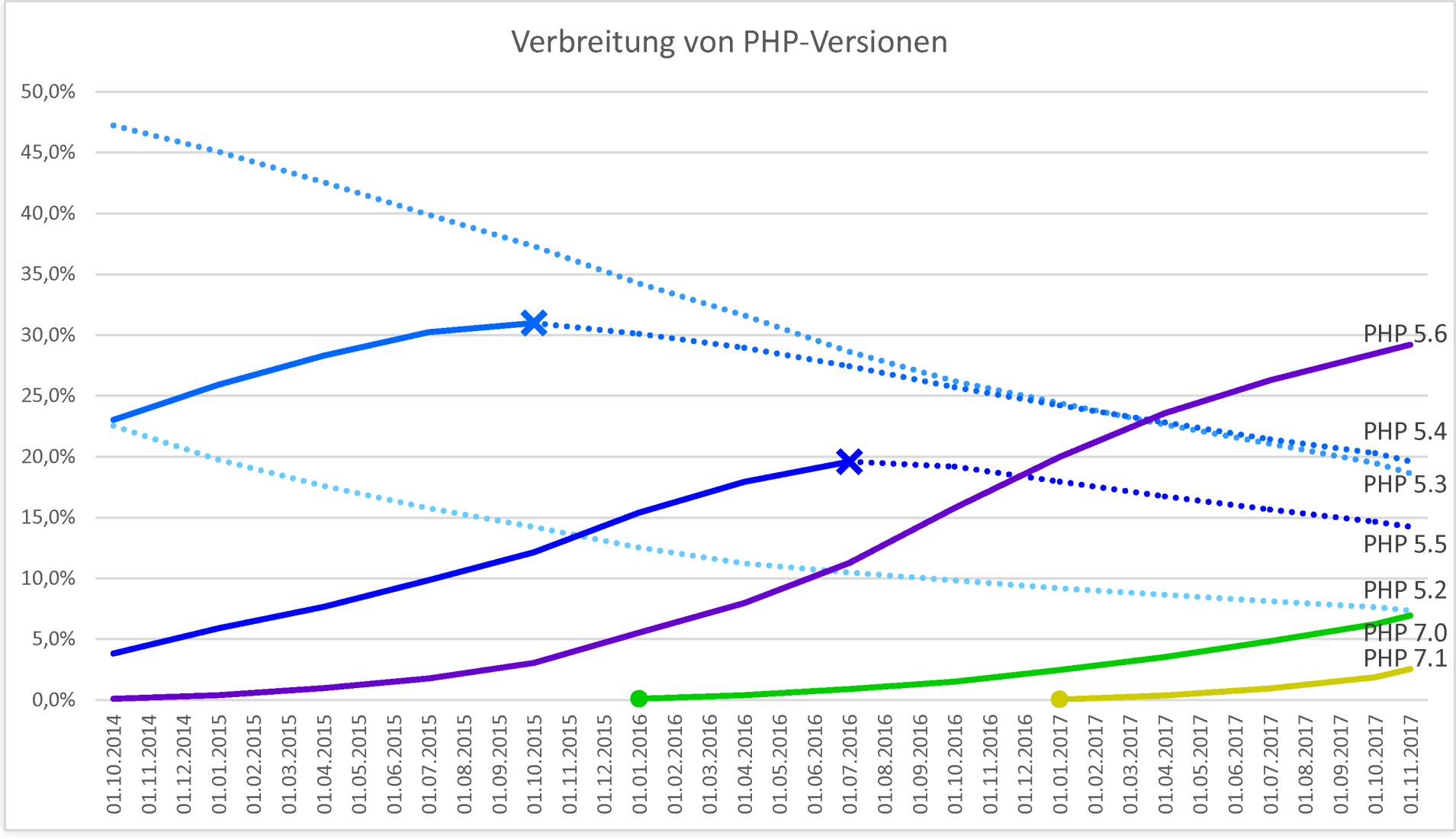
Geschichte, Versionen, Eigenschaften

PHP-Geschichte

- 1995: “**P**ersonal **H**ome **P**age Tools”
  - Satz von Perl-Skripten
- ab PHP/FI 2.0 in C geschrieben
- 1998, PHP 3: “**P**HP: **H**ypertext **P**reprocessor”
  - um Module erweiterbar
- 2000, PHP 4: neue Engine (Zend)
- 2004, PHP 5: neue Engine (Zend 2)
  - bessere Unterstützung von Objekt-Orientierung
- 2015, PHP 7: neue Engine (Zend 3)
  - enorme Spracherweiterung

Quelle: http://php.net/manual/de/history.php.php

PHP-Versionen: 5.6, 7.0 oder 7.1?



Quelle: https://w3techs.com/technologies/history_details/pl-php/all/q

PHP-Laufzeitumgebung

- Verschiedene Umgebungen:
  - **Webbasiert**: Anbindung an Webserver via
    - **SAPI**-Modul: direkte Einbindung in Webserver-Prozess
    - **CGI**: Start von PHP-Interpreter für jede Anfrage
  - **Standalone**: Aufruf über Kommandozeile (CLI)
- Verschiedene Konfigurationsdateien, z.B.:

```
/etc/php5/apache2/php.ini # SAPI
/etc/php5/cgi/php.ini     # CGI
/etc/php5/cli/php.ini     # CLI
```

Quelle: http://php.net/manual/en/install.general.php

PHP-Spracheigenschaften

- Sprachliche Wurzeln:
  - C, C++ (`/* comment */`, `printf("Hallo Welt");`, …)
  - Perl (`# comment`, `echo "Hallo Welt";`, …)
- Interpretierte Sprache
- Typsystem:
  - dynamisch (anders als Java oder C)
  - schwach (anders als Python oder Ruby)
  - “graduell”: implizit, an manchen Stellen auch explizit
- Paradigmen:
  - Imperativ und objekt-orientiert

PHP im Backend

… also angebunden an einen Webserver

Ablauf

1. Webserver erhält HTTP-Request, z.B. `GET /index.php`
   - per Konfiguration: Request-Ziel ist dynamische Ressource
     - d.h. nicht Dateiinhalt auslesen, sondern ausführen
2. Webserver ruft PHP-Interpreter auf
   - (via SAPI-Modul oder CGI, spielt hier große keine Rolle)
   - mit Datei `index.php` als “Eingabe”
3. Interpreter führt PHP-Skript `index.php` aus
   - dieses erzeugt i.d.R. eine textuelle Ausgabe
4. Webserver sendet Ausgabe als HTTP-Reponse an Client
   - üblicherweise mit `Content-type: text/html`

Einfaches Beispiel

- Eine PHP-Datei:

```
<html>
<body>
  <h1><?php echo "Hallo, Welt!" ?></h1>
</body>
</html>
```

- Interpreter wertet nur Code zwischen `<?php` und `?>` aus
  - Rest wird eins-zu-eins beibehalten
- Antwort an Browser:

```
<html>
<body>
  <h1>Hallo, Welt!</h1>
</body>
</html>
```

Einfaches Beispiel: Alternative

- PHP-Datei mit gleichem Effekt:

```php
<?php
echo "<html>";
echo "<body>";
echo "<h1>Hallo, Welt!</h1>";
echo "</body>";
echo "</html>";
?>
```

- Ausgabe:

```
<html><body><h1>Hallo, Welt!</h1></body></html>
```

  - `echo` macht selbst keinen Zeilenumbruch
    - das ginge mit `echo "<html>\n";`

Syntax und Datentypen

Syntax

```php
/* Kommentar */
function gruessen($name) {
    echo "Hallo, $name\n";
}
$wen = 'Peter'; // anderer Kommentar
gruessen($wen); # noch einer
```

- Variablen erkennt man am `$`
- Zeilenenden mit Semikolon
- Blöcke mit geschweiften Klammern
- Funktionaufrufe mit runden Klammern
  - `echo` ist keine Funktion, sondern ein Sprachkonstrukt

Kontrollstrukturen

```
Standard-Syntax:
<?php
  if ($a > $b)      { echo "a is bigger than b";}
  elseif ($a == $b) { echo "a is equal to b"; }
  else              { echo "a is smaller than b"; }
?>

Alternative Syntax:
<?php if ($a == 5): ?>
  A is equal to 5
<?php endif; ?>
```

Quelle: http://php.net/manual/de/control-structures.alternative-syntax.php

Schleifen

```php
// for
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}

// while
$i = 1;
while ($i <= 10) {
    echo $i++;
}

// do-while
$i = 1;
do {
    echo $i++;
} while ($i <= 10);
```

Quelle: http://php.net/manual/de/language.control-structures.php

Fallunterscheidungen

```php
switch ($i) { // switch funktioniert mit Zahlen und Strings
    case "apple":
        echo "i is apple";
        break;
    case "bar":
        echo "i is bar";
        break;
    case "cake":
        echo "i is cake";
        break;
}
```

Quelle: http://php.net/manual/de/control-structures.switch.php

Datentypen

- Primitive Datentypen: boolean, integer, float, string
- Arrays
- Objekte
- callable und iterable
- resource
- NULL

Quelle: http://php.net/manual/de/language.types.intro.php

Type Juggling

```php
$foo = "0";  // $foo is string (ASCII 48)
$foo += 2;   // $foo is now an integer (2)
$foo = $foo + 1.3;  // $foo is now a float (3.3)
$foo = 5 + "10 Little Piggies"; // $foo is integer (15)
$foo = 5 + "10 Small Pigs";     // $foo is integer (15)
```

- Typ-schwacher (`==`) und Typ-starker Vergleich (`===`)
  - Details: http://php.net/manual/de/types.comparisons.php

Quelle: http://php.net/manual/de/language.types.type-juggling.php

Strings

```php
$fruit = 'apple';

// He drank some
// 'apple' juice.
echo "He drank some \n '$fruit' juice.";

// He drank some \n '$fruit' juice.
echo 'He drank some \n \'$fruit\' juice.';
```

- Einfache Anführungszeichen `'`:
  - Variablen und Sonderzeichen wie `\n` werden nicht ausgewertet
- Doppelte Anführungszeichen `"`:
  - Variablen und Sonderzeichen werden ausgewertet
- Details:
  - http://php.net/manual/en/language.types.string.php

Arrays

Die Stärke dynamischer Sprachen

Über Arrays iterieren

- PHP-Code

```php
$names = array("Peter", "Maria", "Klara", "Klaus");
// Array benutzen
echo "<ul>";
for ($i = 0; $i < count($names); $i++) {
  echo "<li>$names[$i]</li>"; // Beachte: " nicht '
}
echo "</ul>";
```

- HTML-Ausgabe

```html
<ul><li>Peter</li><li>Maria</li><li>Klara</li><li>Klaus</li></ul>
```

Numerische vs. Assoziative Arrays

```php
$names = array("Peter", "Maria", "Klara", "Klaus");
$auto = array("farbe" => "rot", "tueren" => 5, "halter" => "Klara");
print_r($names);
print_r($auto);
```

```
Array
(
    [0] => Peter
    [1] => Maria
    [2] => Klara
    [3] => Klaus
)
Array
(
    [farbe] => rot
    [tueren] => 5
    [halter] => Klara
)
```

Gemischte Arrays (`print_r`)

```php
$arr = array("Peter", "farbe" => "rot", 5, 45.6, 7 => true);
print_r($arr);
```

```
Array
(
    [0] => Peter
    [farbe] => rot
    [1] => 5
    [2] => 45.6
    [7] => 1
)
```

Gemischte Arrays (`var_dump`)

```php
$arr = array("Peter", "farbe" => "rot", 5, 45.6, 7 => true);
var_dump($arr);
```

```
array(5) {
  [0]=>
  string(5) "Peter"
  ["farbe"]=>
  string(3) "rot"
  [1]=>
  int(5)
  [2]=>
  float(45.6)
  [7]=>
  bool(true)
}
```

Arrays lesen und schreiben

```php
$arr = array("Peter", "Klara");
$arr[] = "Klaus";  // $arr = array("Peter", "Klara", "Klaus");

$arr[2] = "Mia";   // $arr = array("Peter", "Klara", "Mia");

$arr[4] = "Klaus";
//$arr = array(0 => "Peter", 1=> "Klara", 2 => "Mia", 4 => "Klaus");
```

Arrays schachteln, `foreach`

```php
$students = array(array('name' => 'Klara', 'id' => 's00112233'),
                  array('name' => 'Peter', 'id' => 's11223344'));

// Seit PHP 5.4 geht's auch kürzer
$students = [['name' => 'Klara', 'id' => 's00112233'],
             ['name' => 'Peter', 'id' => 's11223344']];

foreach($students as $stud) {
    echo "name: {$stud['name']} "; // { } machen die variablen
    echo "({$stud['id']})\n";      //     Teile explizit
}
```

```
name: Klara (s00112233)
name: Peter (s11223344)
```

Einige Array-Funktionen

```php
$input  = array(4, 4, 3, 4, 3, 3, 5);
$unique = array_unique($input);   // = array(4, 3, 5)
$count  = count($input);          // = 7
$hasOne = in_array(1, $input);    // = false
$sum    = array_sum($input);      // = 26
$five  = array_search(5, $input); // = 6
$six   = array_search(6, $input); // = false

$auto = array("farbe" => "rot", "tueren" => 5, "halter" => "Klara");
$vals = array_values($auto); // array("rot", 5, "Klara");
$keys = array_keys($auto);  // array("farbe", "tueren", "halter");
$new  = array_combine($keys, $vals); // $auto umständlich kopiert
```

Arrays sortieren

```php
$input = array(4, 4, 3, 4, 3, 3, 5);
sort($input);
// Achtung: sort verändert den übergebenen Parameter direkt
// $input == array(3, 3, 3, 4, 4, 4, 5);

rsort($input);
// $input == array(5, 4, 4, 4, 3, 3, 3);
```

Arrays sortieren: Schlüssel

- `sort`/`rsort` sortieren nach Werten & verwerfen Schlüssel

```php
$students = array('s11223344' => 'Peter',
                  's00112233' => 'Klara',
                  's22334455' => 'Lisa');
sort($students);
print_r($students);
```

```
Array
(
  [0] => Klara
  [1] => Lisa
  [2] => Peter
)
```

Arrays sortieren: Schlüssel

- `asort`/`arsort` bewahren Schlüssel (“Assoziation”)

```php
$students = array('s11223344' => 'Peter',
                  's00112233' => 'Klara',
                  's22334455' => 'Lisa');
asort($students);
print_r($students);
```

```
Array
(
  [s00112233] => Klara
  [s22334455] => Lisa
  [s11223344] => Peter
)
```

Arrays sortieren: Schlüssel

- `ksort`/`krsort` sortieren nach Schlüsseln (“Keys”)

```php
$students = array('s11223344' => 'Peter',
                  's00112233' => 'Klara',
                  's22334455' => 'Lisa');
ksort($students);
print_r($students);
```

```
Array
(
  [s00112233] => Klara
  [s11223344] => Peter
  [s22334455] => Lisa
)
```

13(!) Array-Sortierfunktionen

Funktionsname	Sortiert nach	Pflegt Schlüssel Assoziation	Art der Sortierung
array_multisort()	Wert	Assoziativ: Ja, Numerisch: Nein	Erste Array- oder Sortieroptionen
asort()	Wert	Ja	Aufsteigend
arsort()	Wert	Ja	Absteigend
krsort()	Schlüssel	Ja	Absteigend
ksort()	Schlüssel	Ja	Aufsteigend
natcasesort()	Wert	Ja	Natürlich, Beachtet Groß-/Kleinschreibung
natsort()	Wert	Ja	Natürlich
rsort()	Wert	Nein	Absteigend
shuffle()	Wert	Nein	Zufällig
sort()	Wert	Nein	Aufsteigend
uasort()	Wert	Ja	Benutzerdefiniert
uksort()	Schlüssel	Ja	Benutzerdefiniert
usort()	Wert	Nein	Benutzerdefiniert

Quelle: http://php.net/manual/de/array.sorting.php

Weitere Array-Funktionen

- Arrays: zentraler Datentyp in Skriptsprachen, daher →
- PHP kennt fast 80 Array-Funktionen:
  - http://php.net/manual/de/ref.array.php
- Mächtig sind z.B. `array_map` und `array_filter`

```
array_map(function($v){return $v*2;}, [1,2,3]); // = [2,4,6]
array_filter([1,2,3], function($v){return $v%2;}); // = [1,3]
```

  - hier mit anonymen Funktionen
  - alternativ: Name der Funktion als String übergeben

```
$allTrimmed = array_map('trim', $stringArray);
```

String-Funktionen

Einige String-Funktionen

- Teilstring: `substr`

```php
$rest = substr('abcdef', 1);     // "bcdef"
$rest = substr('abcdef', 1, 3);  // "bcd"
$rest = substr('abcdef', 0, 4);  // "abcd"
$rest = substr('abcdef', 0, 8);  // "abcdef"
$rest = substr('abcdef', -1, 1); // "f"
```

- Stringsuche: `strpos`

```php
$pos = strpos('abc', 'a');  // 0
$pos = strpos('abc', 'bc'); // 1
$pos = strpos('abc', 'd');  // FALSE
```

Quelle: http://php.net/manual/de/ref.strings.php

Einige weitere String-Funktionen

- Aufteilen: `explode`

```php
$arr = explode('.', '192.168.0.1'); // array('192','168','0','1')
```

- Reguläre Ausdrücke: `preg_*`

```php
$is_jpg = preg_match('/\.jpe?g$/i', 'batman.jpg'); // true
$is_jpg = preg_match('/\.jpe?g$/i', 'a.jpg.exe'); // false
```

  - Sehr mächtige Funktionen!
    - http://php.net/manual/de/ref.pcre.php

- Formatieren: `sprintf`

```php
$a = sprintf("%0.2f €", 12);    // "12.00 €"
$b = sprintf("%0.2f €", 7.23);  // "7.23 €"
$c = sprintf("%0.2f €", 5.875); // "5.88 €"
```

Quelle: http://php.net/manual/de/ref.strings.php

Skripte schreiben

Wiederverwendung, Scopes, und Parameter

Code-Wiederverwendung

- PHP-Interpreter arbeitet auf *einer* PHP-Datei
  - weitere Dateien können eingebunden werden
- Datei `index.php`:

```
<html><body>
<?php include 'navi.php'; ?>
<section><h1>Hallo Welt</h1></section>
</body></html>
```

- Datei: `navi.php`:

```
<nav><ol><li>...</li></ol></nav>
```

- Es gibt `include`, `require`, `include_once` und `require_once`
  - http://php.net/manual/de/function.include.php

Funktionsdefinitionen

- Funktionen können überall definiert werden
  - Definierte Funktionen können nicht mehr verändert werden
    - (anders als in JavaScript)
- Man kann prüfen, ob eine Funktion bereits existiert:

```php
// hier kann man do_awesome_stuff() evtl. noch nicht aufrufen
if (!function_exists('do_awesome_stuff')) {
  function do_awesome_stuff() {
    /* ... */
  }
}
// hier kann man do_awesome_stuff() in jedem Fall aufrufen
```

Quelle: http://php.net/manual/de/functions.user-defined.php

Scopes (Gültigkeitsbereiche)

- **Global**: Funktionen und Variablen, die außerhalb von Funktionen definiert sind
  - auch Code-Teile, die per `include` eingebunden wurden
- **In Funktionen**:
  - für Zugriff auf globale Variablen: mit `global` sichtbar machen

```php
$a = 1;
function foo($b) {
    return $a+$b; // "Notice: Undefined variable: a"
}
function bar($b) {
    global $a;
    return $a+$b;
}
foo(2); // 2
bar(2); // 3
```

Quelle: http://php.net/language.variables.scope

Parameterbehandlung

- Änderungen an Parametern nur *in* Funktion sichtbar
  - (außer: Eigenschaften von Objekten ändern)
    - (wie in JavaScript und Java)
- Aber: Übergeben von Referenzen ist *möglich*
  - (nicht wie bei JavaScript und Java)
  - Änderungen sind dann auch außerhalb sichtbar
  - Kennzeichnung mit `&` (Und-Zeichen)
- Beispiel folgt …

JavaScript vs. PHP

```javascript
function changeStuff(a, b, c) {
  a = a * 10;
  b.item = "changed";
  c = {item: "changed"};
}

var num = 10;
var obj1 = {item: "unchanged"};
var obj2 = {item: "unchanged"};
var obj3 = obj1;

changeStuff(num, obj1, obj2);

// num == 10
// obj1.item == "changed"
// obj2.item == "unchanged"
// obj3.item == "changed"
```

```php
function changeStuff($a, $b, $c) {
  $a = $a * 10;
  $b->item = 'changed';
  $c=(object)['item'=>'changed'];
}

$num = 10;
$o1=(object)['item'=>'unchanged'];
$o2=(object)['item'=>'unchanged'];
$o3=$o1;

changeStuff($num, $o1, $o2);

// $num == 10
// $o1->item == "changed"
// $o2->item == "unchanged"
// $o3->item == "changed"
```

PHP: Ohne vs. mit Referenz (&)

```php
// Signatur vorher:
function changeStuff($a, $b, $c) {
  $a = $a * 10;
  $b->item = 'changed';
  $c=(object)['item'=>'changed'];
}

$num = 10;
$o1=(object)['item'=>'unchanged'];
$o2=(object)['item'=>'unchanged'];
$o3=$o1;

changeStuff($num, $o1, $o2);

// $num == 10
// $o1->item == "changed"
// $o2->item == "unchanged"
// $o3->item == "changed"
```

```php
// Signatur mit &:
function changeStuff(&$a, $b, &$c) {
  $a = $a * 10;
  $b->item = 'changed';
  $c=(object)['item'=>'changed'];
}

$num = 10;
$o1=(object)['item'=>'unchanged'];
$o2=(object)['item'=>'unchanged'];
$o3=$o1;

changeStuff($num, $o1, $o2);

// $num == 100
// $o1->item == "changed"
// $o2->item == "changed"
// $o3->item == "changed"
```

Ok, aber …

Warum?

Wofür braucht man das?

By Reference – Anwendungsfälle

1. Primär: Effekt auf Parameter; Rückgabewert selten relevant

```
bool sort(array &$array)
```

   - Effekt: Sortiertes Array
   - Rückgabewert: Erfolg
     - http://php.net/manual/de/function.sort.php

2. “Mehrere” Rückgabewerte

```
int preg_match_all($pattern, $subject, array &$matches)
```

   - Primärer Rückgabewert: Anzahl Treffer für Regex
   - Sekundärer Rückgabewert: Die konkreten Treffer
     - http://php.net/manual/de/function.preg-match-all.php

Wie entwickelt man jetzt

Webanwendungen

mit PHP?

Aufbau von Webanwendungen

Von der Website zur Anwendung

Szenario 1: Statische Website

- Zusammenbau einer Supero-artigen Website
  - Einheitliche Navigation?

```php
include 'navi.php';
```

  - Helden-Tabelle?

```php
$heros = [['name' => 'Batman', 'img' => 'batman.jpg'],
          ['name' => 'Robin', 'img' => 'robin02.jpg']];
// ...
echo "<table>";
foreach($heros as $held) { echo "<tr>"; /* ... */ };
echo "</table>";
```

  - PHP als Code-Generator für statische Webseiten

Szenario 2: Kontaktforumlar

- Behandlung einzelner Anfragen
  - `kontakt.htm`

```
<form action='senden.php' method='post'>
 <textarea name='text'></textarea>
 <input type='submit'>
</form>
```

  - `senden.php`

```
<?php // Formular-Eingaben behandeln ?>
<p>Vielen Dank für Ihre Nachricht!</p>
```

Variablen über HTTP

- Erinnerung: HTTP kennt GET- und POST-Variablen
  - GET: Variablen in URL (`query`-Teil)

```
GET /index.php?name=Klaus HTTP/1.1
```

  - POST: Variablen in HTTP-Request-Body

```
POST /index.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 10

name=Klaus
```

- PHP kann mit beidem umgehen

Eingabe-Parameter

- “superglobale” Arrays `$_GET` und `$_POST`
  - sind auch ohne `global` in allen Scopes verfügbar
  - `kontakt.htm`

```
<form action='senden.php' method='post'>
  <textarea name='text'></textarea>
  <input type='submit'>
</form>
```

  - `senden.php`

```
<?php $sent = mail(
        'info@example.org',                 // Empfänger
        'Kontakt',                          // Betreff
        $_POST['text'],                     // Inhalt
        'From: formular@example.org');      // E-Mail-Header
if($sent) : ?>
  <p>Vielen Dank für Ihre Nachricht!</p>
<?php endif ; ?>
```

Weitere superglobale Variablen

- `$_COOKIE`: vom Client mitgesendete Cookies
  - (Cookies setzen geht mit setcookie())
- `$_REQUEST`: Vereinigung von `$_GET`, `$_POST` und `$_COOKIE`
- `$_FILES`: vom Client hochgeladene Dateien
  - Behandlung von Fileuploads
- `$_SESSION`: Sitzungsdaten
- `$_ENV`: Umgebungsvariablen
- `$_SERVER`: Aufrufparameter
  - z.B. CGI-Infos, CLI-Argumente, ...

Quelle: http://php.net/manual/de/language.variables.superglobals.php

Kleine Skripte: Taschenrechner

- PHP-Skript erwartet Werte für `a`, `b` und `op`

```php
$a = $_GET['a']; $b = $_GET['b'];
switch($_GET['op']) {
 case 'add':
   echo "$a + $b = <b>" . ($a+$b) . "</b>";
   break;
 default: echo "not implemented yet";
}
```

- Eingabe der Werte über URL

```
GET /calc.php?op=add&a=5&b=12 HTTP/1.1
```

- Ausgabe des Skripts

```
5 + 12 = <b>17</b>
```

Szenarien

1. Statische Seite, HTML-Generierung ✓
2. Einfache Ausführung, Skripte ✓
3. Anwendungen
    - Nicht nur *Eingabe*, *Verarbeitung*, *Ausgabe*
    - Sondern: *Interaktion* mit Nutzer/in

Szenario 3: Webanwendung

- Interaktive Anwendung hat verschiedene Zustände
  - diese müssen abgebildet werden
- **Beispiel**: Anwendung, die Nutzer/in grüßt
  - Wenn sie den Namen kennt, dann mit Namen
  - Nutzer/in kann einen anderen Namen eingeben
- Zustände der Beispielanwendung:
  1. Anwendung kennt keinen Namen (initial)
  2. Anwendung kennt Namen
- Zustände: hier “Actions”

Actions repräsentieren

- **Idee 1**: Eine Datei pro *Action*
  - *Initial*: `index.php`

```php
echo "<p>Wie soll ich dich nennen?</p>";
include 'greet-form.php';
```

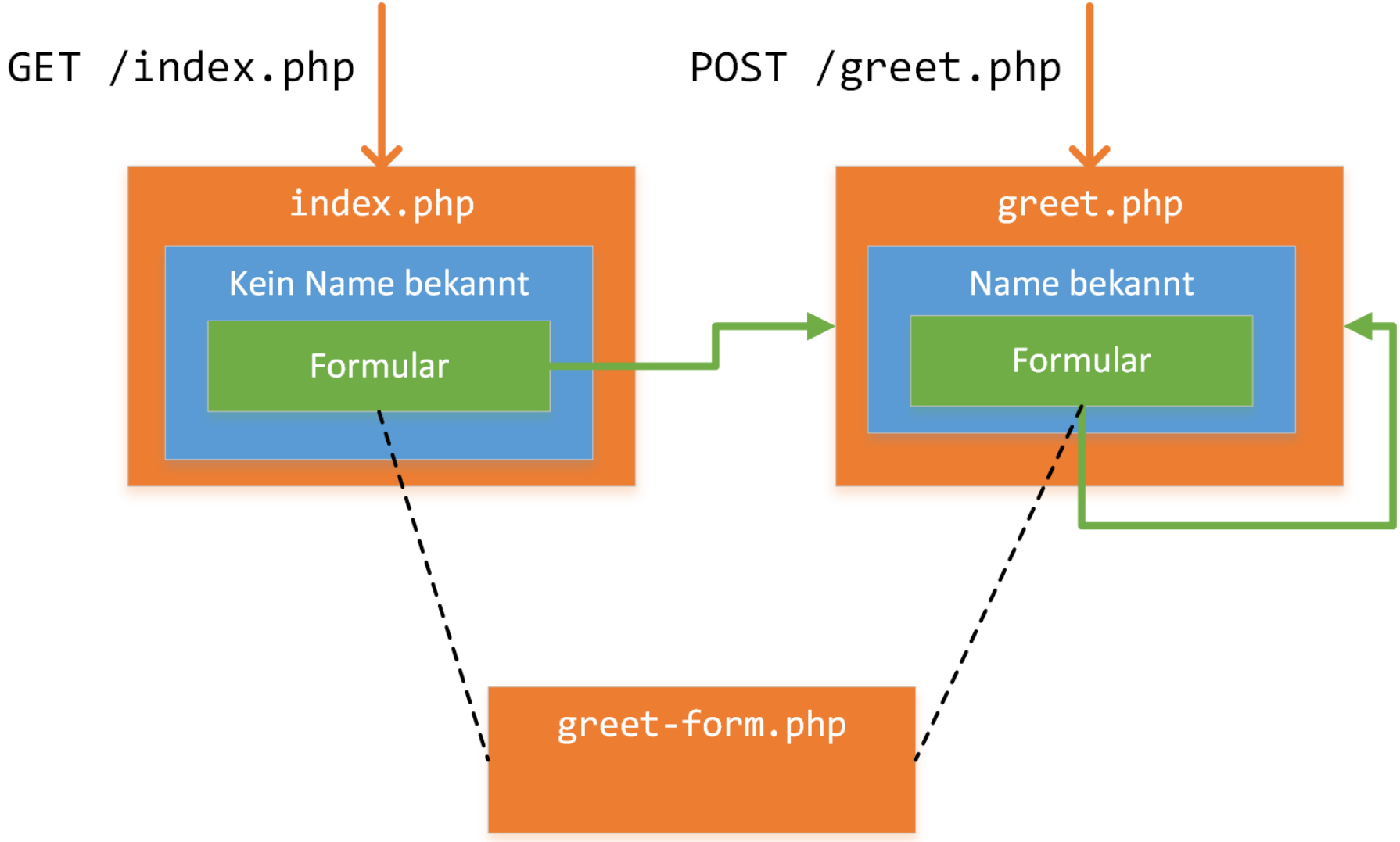
  - *Name bekannt*: `greet.php`

```php
echo "<p>Hallo {$_POST['name']}! Oder soll ich dich anders nennen?</p>";
include 'greet-form.php';
```

  - ausgelagert: `greet-form.php`

```php
<form method="post" action="greet.php">
  Name: <input name="name"><input type="submit">
</form>
```

Idee 1: Grafisch



Actions repräsentieren

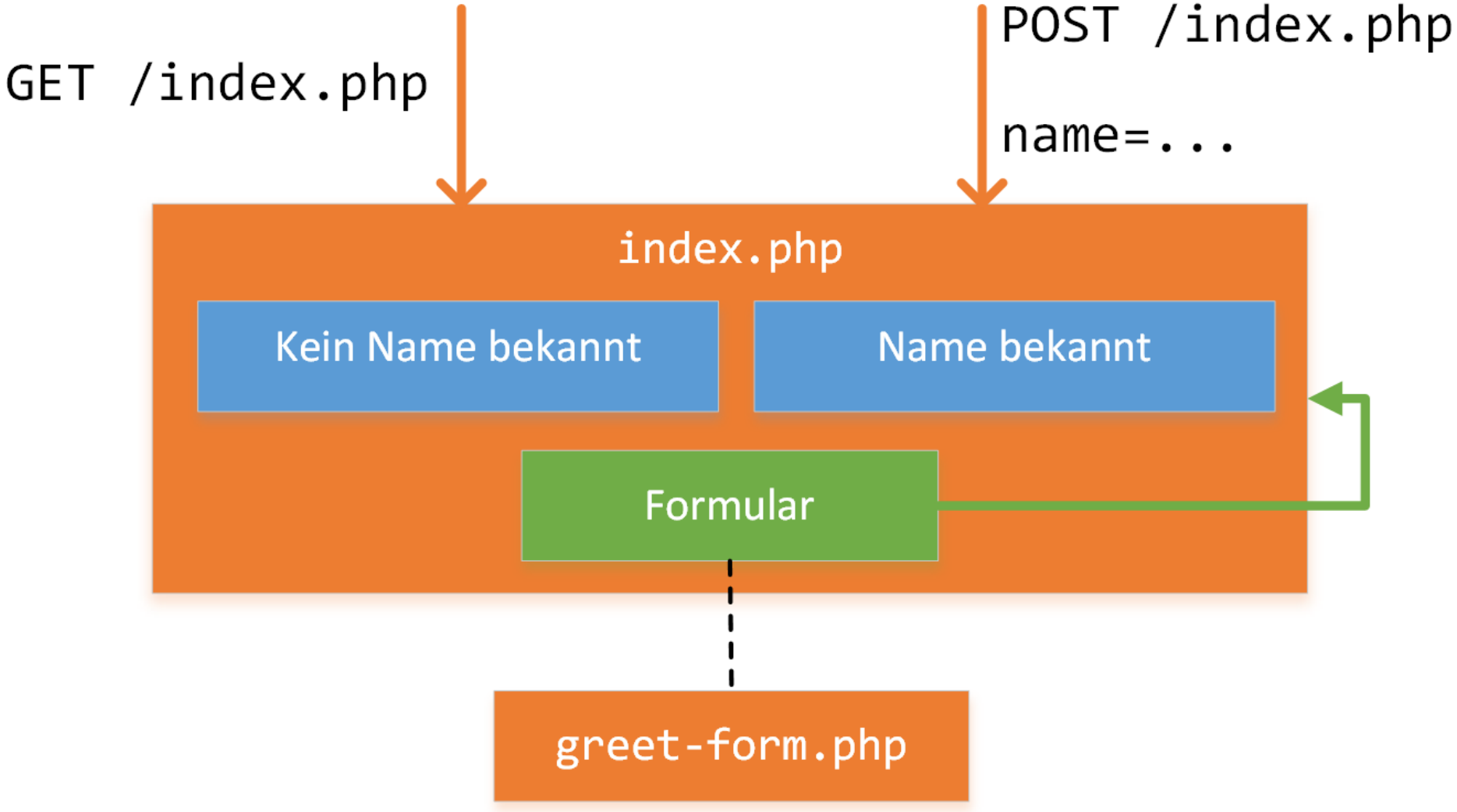
- **Idee 2**: Actions in *einer* Datei unterscheiden
  - `index.php`

```php
if (isset($_POST['name'])) {
  echo "<p>Hallo {$_POST['name']}! Oder soll ich dich anders nennen?</p>";
} else {
  echo "<p>Wie soll ich dich nennen?</p>";
}
include 'greet-form.php';
```

  - `greet-form.php`

```php
<form method="post" action="index.php">
  Name: <input name="name"><input type="submit">
</form>
```

Idee 2: Grafisch



Verarbeitung von Eingabewerten

- Ist Variable gesetzt?: `isset($_POST['name'])`
  - `if($_POST['name'] != null)` kann Hinweismeldung erzeugen, wenn `name` nicht gesetzt ist
- Vorhandene Variable muss keinen gültigen Wert haben, typische Prüfungen und Vorverarbeitungen:

```php
$name  = isset($_POST['name'])  ? trim($_POST['name'])    : "";
$alter = isset($_POST['alter']) ? intval($_POST['alter']) : 0;
```

- Seit PHP 5.2 erleichtert/ergänzt mit Filterfunktionen
  - http://php.net/manual/de/ref.filter.php

Idee 2 mit Vorverarbeitung

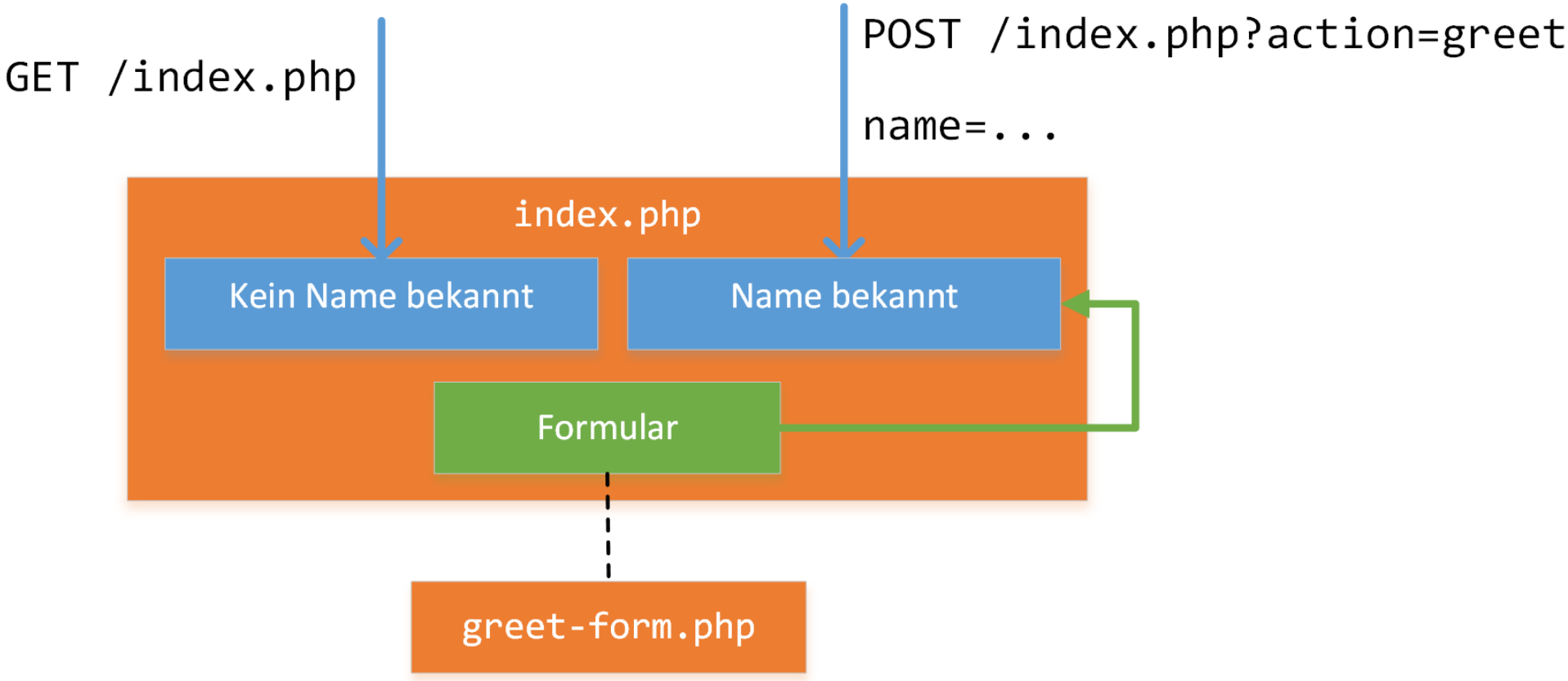
```php
$name = isset($_POST['name']) ? trim($_POST['name']) : "";
if ($name != "") {
  echo "<p>Hallo $name! Oder soll ich dich anders nennen?</p>";
} else {
  echo "<p>Wie soll ich dich nennen?</p>";
}
include 'greet-form.php';
```

- Entwurfsqualität?
  - Zustände implizit: Entscheidung hängt an Geschäfts-Variablen
    - Was, wenn zwei oder mehr Variablen beteiligt sind?
  - Fehlerbehandlung und Zustandsbestimmung vermischt

Idee 2: Explizite Zustände

```php
$action = isset($_GET['action']) ? trim($_GET['action']) : "init";
switch ($action) {
  case "init":
    echo "<p>Wie soll ich dich nennen?</p>";
    break;
  case "greet":
    $name = isset($_POST['name']) ? trim($_POST['name']) : "";
    if ($name != "") {
      echo "<p>Hallo $name! Oder soll ich dich anders nennen?</p>";
      break;
    }
  default:
    echo "<p>Nanu? Ich habe dich nicht verstanden</p>";
}
include 'greet-form.php'; // <form action="index.php?action=greet" ..
```

Idee 2 (explizit): Grafisch



Entwicklung von Webanwendungen

- Heute nur sehr grob besprochen
  - Konzeptionell: Zerlegen der Nutzerinteraktionen in *Actions*
  - Entscheidung für *Entwurfsstil*: Wie viele Actions pro PHP-Datei?
    - Falls mehrere pro Datei: Wie unterscheiden?
- Folgende Einheiten (besonders übernächste): Im Detail

Szenarien

1. Statische Seite, HTML-Generierung: ✓
2. Einfache Ausführung, Skripte: ✓
3. Anwendungen: (✓)

Vielseitigkeit von PHP

Ausgabe-Manipulation

- `header()` zum Setzen von HTTP-Header-Zeilen, z.B.
  - Status-Code

```
header("HTTP/1.0 404 Not Found");
```

  - Umleitung

```
header("Location: http://www.example.com/");
```

  - Dateitypen

```
header('Content-Type: text/plain');
```

  - Downloads

```
header('Content-Type: application/pdf');
header('Content-Disposition: attachment; filename="downloaded.pdf"');
```

Quelle: http://php.net/manual/de/function.header.php

Dateisystem-Zugriffe

- Low-level (C-artig, Handle-basiert)
  - `opendir()`, `readdir()`, `closedir()`
    - http://php.net/manual/de/ref.dir.php
  - `fopen()`, `fread()`, `fwrite()`, `fclose()`
    - erlaubt byteweisen Zugriff
- High-level
  - Dateiinhalt zeilenweise in Array: `file()`
  - Schreiben und Lesen: `file_get_contents()`/`file_put_contents()`

Quelle: http://php.net/manual/de/ref.filesystem.php

Weitere Extensions von PHP

- Verwalten von Sessions
  - `$_SESSION['name'] = "Peter";` … `echo $_SESSION['name'];`
- Umgang mit ZIP-Dateien
- Datums- und Zeitangaben
- XML-Verarbeitung
- Datenbankanbindungen
- Audio-Formate
- Bild-Verarbeitung
- mehr: http://php.net/manual/de/funcref.php

Ausführung & Debugging

Ausführung von PHP-Skripten

- Webserver (lokal XAMPP, Heroku, HTW-Server, …)
  - PHP-Dateien in gleiches Verzeichnis wie HTML-Dateien
  - Aufruf im Webbrowser via HTTP (nicht `file:///...`)
- Entwicklungsserver (seit PHP 5.4)
  - Mini-Webserver, im aktuellen Verzeichnis

```
$> php -S localhost:8000 # oder ein anderer Port
```

- Von der Kommandozeile (Datei)

```
$> php index.php # oder eine andere PHP-Datei
```

- Von der Kommandozeile (Interaktiv, seit PHP 5.1)

```
$> php -a
```

Informationen und Konfiguration

- `phpinfo()` erzeugt Übersicht über alle Einstellungen
  - und die geladenen Erweiterungen
  - (Ausgabe: HTML bei SAPI/CGI, Text bei CLI)
- Fehler- und Hinweismeldungen
  - **Entwicklungssystem**: Fehler und Hinweise anzeigen
  - **Produktivsystem**: Keine Meldungen anzeigen, aber loggen
  - (z.B.
    - `error_reporting` (Log-Level): `E_ALL` vs. `E_ERROR | E_WARNING`
    - `display_errors` (Anzeigen): `1` vs. `0`
    - `log_errors` (Loggen): `0` vs. `1`)
  - Sie sehen nur weiße Seite ohne Inhalt & ohne Meldung?
    - Könnte an `display_errors: 0` liegen
  - http://php.net/manual/de/book.errorfunc.php

Debugging

- Simpelste Methode:
  - `print_r()` oder `var_dump()`
    - an beliebiger Stelle im Skript Variablen-Inhalte ausgeben
  - `die()` oder `die("Komme ich bis hier hin?")`
    - Ausführung des Skripts an beliebiger Stelle abbrechen
    - Erscheint die Meldung, ist ihr Skript immerhin bis dorthin gekommen
- Integrierter Debugger: PHPDBG

```
$> phpdbg -h
```

Zusammenfassung

- Kriterien zur Auswahl einer Backend-Technologie
- PHP-Grundlagen
  - Arrays und Array-Funktionen
  - String-Funktionen
- PHP im Webserver-Kontext
  - Generierung von HTML-Code
  - Skripte (Request - Response)
  - Webanwendungen (Interaktionen, Zustände)
- Mehr Funktionalität durch PHP-Erweiterungen
- Konfigurationsinfos und einfaches Debugging

Danke!